L3: Entry 3 of 14 File: USPT Oct 22, 2002

DOCUMENT-IDENTIFIER: US 6470492 B2
TITLE: Low overhead speculative selection of hot traces in a caching dynamic translator

Brief Summary Text (11):
Hot traces can also be constructed indirectly, using branch or basic block profiling (as contrasted with trace profiling, where the profile directly provides trace information). In this scheme, a counter is associated with the taken target of every branch (there are other variations on this, but the overheads are similar). When the caching dynamic translator is interpreting the program code, it increments such a counter each time a taken branch is interpreted. When a counter exceeds a preset threshold, its corresponding block is flagged as hot. These hot blocks can be strung together to create a hot trace. Such a profiling technique has the following shortcomings: 1. A large counter table is required, since the number of distinct blocks executed by a program can be very large. 2. The overhead for trace selection is high. The reason can be intuitively explained: if a trace consists of N blocks, this scheme will have to wait until N counters all exceed their thresholds before they can be strung into a trace. It does not take advantage of the fact that after the first counter gets hot, the next N-1 counters are very likely to get hot in quick succession, making it unnecessary to bother incrementing them and doing the bookkeeping of the past blocks that have just executed.

Detailed Description Text (2):
Referring to FIG. 1, a dynamic translator includes an interpreter 110 that receives an input instruction stream 160. This "interpreter" represents the instruction evaluation engine; it can be implemented in a number of ways (e.g., as a software fetch--decode--eval loop, a just-in-time compiler, or even a hardware CPU).

Detailed Description Text (16):
If the start-of-trace condition is not met, then control re-enters the basic interpreter state and interpretation continues. In this case, there is no need to maintain a counter; a counter increment takes place only if a start-of-trace condition is met. This is in contrast to conventional dynamic translator implementations that have maintained counters for each branch target. In the illustrative embodiment counters are only associated with the address of the backward taken branch targets and with targets of branches that exit the translated code cache; thus, the present invention permits a system to use less counter storage and to incur less counter increment overhead.

Detailed Description Text (17):
If the determination of whether a "start-of-trace" condition exists 230 is that the start-of-trace condition is met, then, if a counter for the target does not exist, one is created or if a counter for the target does exist, that counter is incremented.

Detailed Description Text (21):
The lower the threshold, the less time is spent in the interpreter, and the greater the number of start-of-traces that potentially get hot. This results in a greater number of traces being generated into the code cache (and the more speculative the choice of hot traces), which in turn can increase the pressure on the code cache

resources, and hence the overhead of managing the code cache. On the other hand, the higher the threshold, the greater the interpretive overhead (e.g., allocating and incrementing counters associated with start-of-traces). Thus the choice of threshold has to balance these two forces. It also depends on the actual interpretive and code cache management overheads in the particular implementation. In our specific implementation, where the interpreter was written as a software fetch-decode-eval loop in C, a threshold of 50 was chosen as the best compromise.

Detailed Description Text (30):
The foregoing has described a specific embodiment of the invention. Additional variations will be apparent to those skilled in the art. For example, although the invention has been described in the context of a dynamic translator, it can also be used in other systems that employ interpreters or just-in-time compilers (JITs). Further, the invention could be employed in other systems that emulate any non-native system, such as a simulator. Thus, the invention is not limited to the specific details and illustrative example shown and described in this specification. Rather, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

CLAIMS:

1. In a dynamic translator, a method for selecting hot traces in a program being translated comprising the steps of: (A) dynamically associating counters with addresses in the program being translated that are determined, during program translation and execution, to meet a start-of-trace condition; (B) when an instruction with a corresponding counter is executed, incrementing that counter, and (C) when a counter exceeds a threshold, determining the particular trace (of the possible plurality of traces beginning at that address) that begins at the address corresponding to that counter and is defined by the path of execution taken by the program following that particular execution of that instruction and continuing until an end-of-trace condition is met and identifying that trace as a hot trace.

12. In a dynamic translator having a cache, a method for selecting hot traces comprising the steps of: (A) when a backward branch is taken, if a counter does not exist for the branch target, then creating such a counter, if such a counter does exist, then incrementing the counter; (B) if a counter exceeds a threshold, then storing in the cache a translation of those instructions executed starting at the branch target associated with the counter that exceeded the threshold and continuing until an end-of-trace condition is reached.

Previous Doc Next Doc Go to Doc#